This chapter covers all the interfaces of Selenium WebDriver.

**Recommended Import Style**

The API definitions in this chapter show the absolute location of classes. However, the recommended import style is as given below:

**from** **selenium** **import** webdriver

Then, you can access the classes like this:

webdriver.Firefox

webdriver.FirefoxProfile

webdriver.Chrome

webdriver.ChromeOptions

webdriver.Ie

webdriver.Opera

webdriver.PhantomJS

webdriver.Remote

webdriver.DesiredCapabilities

webdriver.ActionChains

webdriver.TouchActions

webdriver.Proxy

The special keys class (Keys) can be imported like this:

**from** **selenium.webdriver.common.keys** **import** Keys

The exception classes can be imported like this (Replace the TheNameOfTheExceptionClass with the actual class name given below):

**from** **selenium.common.exceptions** **import** [TheNameOfTheExceptionClass]

**Conventions used in the API**

Some attributes are callable (or methods) and others are non-callable (properties). All the callable attributes are ending with round brackets.

Here is an example for property:

* current\_url

URL of the currently loaded page.

Usage:

driver.current\_url

Here is an example of a method:

* close()

Closes the current window.

Usage:

driver.close()

7.1. Exceptions

Exceptions that may happen in all the webdriver code.

*exception*selenium.common.exceptions.**ElementNotInteractableException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidElementStateException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidElementStateException)

Thrown when an element is present in the DOM but interactions with that element will hit another element do to paint order

*exception*selenium.common.exceptions.**ElementNotSelectableException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidElementStateException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidElementStateException)

Thrown when trying to select an unselectable element.

For example, selecting a ‘script’ element.

*exception*selenium.common.exceptions.**ElementNotVisibleException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidElementStateException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidElementStateException)

Thrown when an element is present on the DOM, but it is not visible, and so is not able to be interacted with.

Most commonly encountered when trying to click or read text of an element that is hidden from view.

*exception*selenium.common.exceptions.**ErrorInResponseException**(*response*, *msg*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when an error has occurred on the server side.

This may happen when communicating with the firefox extension or the remote driver server.

*exception*selenium.common.exceptions.**ImeActivationFailedException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when activating an IME engine has failed.

*exception*selenium.common.exceptions.**ImeNotAvailableException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when IME support is not available. This exception is thrown for every IME-related method call if IME support is not available on the machine.

*exception*selenium.common.exceptions.**InvalidArgumentException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

The arguments passed to a command are either invalid or malformed.

*exception*selenium.common.exceptions.**InvalidCookieDomainException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when attempting to add a cookie under a different domain than the current URL.

*exception*selenium.common.exceptions.**InvalidElementStateException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

*exception*selenium.common.exceptions.**InvalidSelectorException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.NoSuchElementException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.NoSuchElementException)

Thrown when the selector which is used to find an element does not return a WebElement. Currently this only happens when the selector is an xpath expression and it is either syntactically invalid (i.e. it is not a xpath expression) or the expression does not select WebElements (e.g. “count(//input)”).

*exception*selenium.common.exceptions.**InvalidSwitchToTargetException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when frame or window target to be switched doesn’t exist.

*exception*selenium.common.exceptions.**MoveTargetOutOfBoundsException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when the target provided to the *ActionsChains* move() method is invalid, i.e. out of document.

*exception*selenium.common.exceptions.**NoAlertPresentException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when switching to no presented alert.

This can be caused by calling an operation on the Alert() class when an alert is not yet on the screen.

*exception*selenium.common.exceptions.**NoSuchAttributeException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when the attribute of element could not be found.

You may want to check if the attribute exists in the particular browser you are testing against. Some browsers may have different property names for the same property. (IE8’s .innerText vs. Firefox .textContent)

*exception*selenium.common.exceptions.**NoSuchElementException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when element could not be found.

If you encounter this exception, you may want to check the following:

* Check your selector used in your find\_by...
* Element may not yet be on the screen at the time of the find operation, (webpage is still loading) see selenium.webdriver.support.wait.WebDriverWait() for how to write a wait wrapper to wait for an element to appear.

*exception*selenium.common.exceptions.**NoSuchFrameException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidSwitchToTargetException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidSwitchToTargetException)

Thrown when frame target to be switched doesn’t exist.

*exception*selenium.common.exceptions.**NoSuchWindowException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.InvalidSwitchToTargetException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.InvalidSwitchToTargetException)

Thrown when window target to be switched doesn’t exist.

To find the current set of active window handles, you can get a list of the active window handles in the following way:

print driver.window\_handles

*exception*selenium.common.exceptions.**RemoteDriverServerException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

*exception*selenium.common.exceptions.**StaleElementReferenceException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a reference to an element is now “stale”.

Stale means the element no longer appears on the DOM of the page.

Possible causes of StaleElementReferenceException include, but not limited to:

* You are no longer on the same page, or the page may have refreshed since the element was located.
* The element may have been removed and re-added to the screen, since it was located. Such as an element being relocated. This can happen typically with a javascript framework when values are updated and the node is rebuilt.
* Element may have been inside an iframe or another context which was refreshed.

*exception*selenium.common.exceptions.**TimeoutException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a command does not complete in enough time.

*exception*selenium.common.exceptions.**UnableToSetCookieException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a driver fails to set a cookie.

*exception*selenium.common.exceptions.**UnexpectedAlertPresentException**(*msg=None*, *screen=None*, *stacktrace=None*, *alert\_text=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when an unexpected alert is appeared.

Usually raised when when an expected modal is blocking webdriver form executing any more commands.

*exception*selenium.common.exceptions.**UnexpectedTagNameException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: [**selenium.common.exceptions.WebDriverException**](http://selenium-python.readthedocs.io/api.html#selenium.common.exceptions.WebDriverException)

Thrown when a support class did not get an expected web element.

*exception*selenium.common.exceptions.**WebDriverException**(*msg=None*, *screen=None*, *stacktrace=None*)

Bases: **exceptions.Exception**

Base webdriver exception.

7.2. Action Chains

The ActionChains implementation,

*class*selenium.webdriver.common.action\_chains.**ActionChains**(*driver*)

Bases: **object**

ActionChains are a way to automate low level interactions such as mouse movements, mouse button actions, key press, and context menu interactions. This is useful for doing more complex actions like hover over and drag and drop.

Generate user actions.

When you call methods for actions on the ActionChains object, the actions are stored in a queue in the ActionChains object. When you call perform(), the events are fired in the order they are queued up.

ActionChains can be used in a chain pattern:

menu = driver.find\_element\_by\_css\_selector(".nav")

hidden\_submenu = driver.find\_element\_by\_css\_selector(".nav #submenu1")

ActionChains(driver).move\_to\_element(menu).click(hidden\_submenu).perform()

Or actions can be queued up one by one, then performed.:

menu = driver.find\_element\_by\_css\_selector(".nav")

hidden\_submenu = driver.find\_element\_by\_css\_selector(".nav #submenu1")

actions = ActionChains(driver)

actions.move\_to\_element(menu)

actions.click(hidden\_submenu)

actions.perform()

Either way, the actions are performed in the order they are called, one after another.

**click**(*on\_element=None*)

Clicks an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to click. If None, clicks on current mouse position. |

**click\_and\_hold**(*on\_element=None*)

Holds down the left mouse button on an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to mouse down. If None, clicks on current mouse position. |

**context\_click**(*on\_element=None*)

Performs a context-click (right click) on an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to context-click. If None, clicks on current mouse position. |

**double\_click**(*on\_element=None*)

Double-clicks an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to double-click. If None, clicks on current mouse position. |

**drag\_and\_drop**(*source*, *target*)

Holds down the left mouse button on the source element,

then moves to the target element and releases the mouse button.

|  |  |
| --- | --- |
| **Args:** | * source: The element to mouse down. * target: The element to mouse up. |

**drag\_and\_drop\_by\_offset**(*source*, *xoffset*, *yoffset*)

Holds down the left mouse button on the source element,

then moves to the target offset and releases the mouse button.

|  |  |
| --- | --- |
| **Args:** | * source: The element to mouse down. * xoffset: X offset to move to. * yoffset: Y offset to move to. |

**key\_down**(*value*, *element=None*)

Sends a key press only, without releasing it.

Should only be used with modifier keys (Control, Alt and Shift).

|  |  |
| --- | --- |
| **Args:** | * value: The modifier key to send. Values are defined in *Keys* class. * element: The element to send keys. If None, sends a key to current focused element. |

Example, pressing ctrl+c:

ActionChains(driver).key\_down(Keys.CONTROL).send\_keys('c').key\_up(Keys.CONTROL).perform()

**key\_up**(*value*, *element=None*)

Releases a modifier key.

|  |  |
| --- | --- |
| **Args:** | * value: The modifier key to send. Values are defined in Keys class. * element: The element to send keys. If None, sends a key to current focused element. |

Example, pressing ctrl+c:

ActionChains(driver).key\_down(Keys.CONTROL).send\_keys('c').key\_up(Keys.CONTROL).perform()

**move\_by\_offset**(*xoffset*, *yoffset*)

Moving the mouse to an offset from current mouse position.

|  |  |
| --- | --- |
| **Args:** | * xoffset: X offset to move to, as a positive or negative integer. * yoffset: Y offset to move to, as a positive or negative integer. |

**move\_to\_element**(*to\_element*)

Moving the mouse to the middle of an element.

|  |  |
| --- | --- |
| **Args:** | * to\_element: The WebElement to move to. |

**move\_to\_element\_with\_offset**(*to\_element*, *xoffset*, *yoffset*)

Move the mouse by an offset of the specified element.

Offsets are relative to the top-left corner of the element.

|  |  |
| --- | --- |
| **Args:** | * to\_element: The WebElement to move to. * xoffset: X offset to move to. * yoffset: Y offset to move to. |

**perform**()

Performs all stored actions.

**release**(*on\_element=None*)

Releasing a held mouse button on an element.

|  |  |
| --- | --- |
| **Args:** | * on\_element: The element to mouse up. If None, releases on current mouse position. |

**reset\_actions**()

Clears actions that are already stored on the remote end.

**send\_keys**(*\*keys\_to\_send*)

Sends keys to current focused element.

|  |  |
| --- | --- |
| **Args:** | * keys\_to\_send: The keys to send. Modifier keys constants can be found in the ‘Keys’ class. |

**send\_keys\_to\_element**(*element*, *\*keys\_to\_send*)

Sends keys to an element.

|  |  |
| --- | --- |
| **Args:** | * element: The element to send keys. * keys\_to\_send: The keys to send. Modifier keys constants can be found in the ‘Keys’ class. |

7.3. Alerts

The Alert implementation.

*class*selenium.webdriver.common.alert.**Alert**(*driver*)

Bases: **object**

Allows to work with alerts.

Use this class to interact with alert prompts. It contains methods for dismissing, accepting, inputting, and getting text from alert prompts.

Accepting / Dismissing alert prompts:

Alert(driver).accept()

Alert(driver).dismiss()

Inputting a value into an alert prompt:

name\_prompt = Alert(driver) name\_prompt.send\_keys(“Willian Shakesphere”) name\_prompt.accept()

Reading a the text of a prompt for verification:

alert\_text = Alert(driver).text self.assertEqual(“Do you wish to quit?”, alert\_text)

**accept**()

Accepts the alert available.

Usage:: Alert(driver).accept() # Confirm a alert dialog.

**authenticate**(*username*, *password*)

Send the username / password to an Authenticated dialog (like with Basic HTTP Auth). Implicitly ‘clicks ok’

Usage:: driver.switch\_to.alert.authenticate(‘cheese’, ‘secretGouda’)

|  |  |
| --- | --- |
| **Args:** | -username: string to be set in the username section of the dialog -password: string to be set in the password section of the dialog |

**dismiss**()

Dismisses the alert available.

**send\_keys**(*keysToSend*)

Send Keys to the Alert.

|  |  |
| --- | --- |
| **Args:** | * keysToSend: The text to be sent to Alert. |

**text**

Gets the text of the Alert.

7.4. Special Keys

The Keys implementation.

*class*selenium.webdriver.common.keys.**Keys**

Bases: **object**

Set of special keys codes.

**ADD***= u'\ue025'*

**ALT***= u'\ue00a'*

**ARROW\_DOWN***= u'\ue015'*

**ARROW\_LEFT***= u'\ue012'*

**ARROW\_RIGHT***= u'\ue014'*

**ARROW\_UP***= u'\ue013'*

**BACKSPACE***= u'\ue003'*

**BACK\_SPACE***= u'\ue003'*

**CANCEL***= u'\ue001'*

**CLEAR***= u'\ue005'*

**COMMAND***= u'\ue03d'*

**CONTROL***= u'\ue009'*

**DECIMAL***= u'\ue028'*

**DELETE***= u'\ue017'*

**DIVIDE***= u'\ue029'*

**DOWN***= u'\ue015'*

**END***= u'\ue010'*

**ENTER***= u'\ue007'*

**EQUALS***= u'\ue019'*

**ESCAPE***= u'\ue00c'*

**F1***= u'\ue031'*

**F10***= u'\ue03a'*

**F11***= u'\ue03b'*

**F12***= u'\ue03c'*

**F2***= u'\ue032'*

**F3***= u'\ue033'*

**F4***= u'\ue034'*

**F5***= u'\ue035'*

**F6***= u'\ue036'*

**F7***= u'\ue037'*

**F8***= u'\ue038'*

**F9***= u'\ue039'*

**HELP***= u'\ue002'*

**HOME***= u'\ue011'*

**INSERT***= u'\ue016'*

**LEFT***= u'\ue012'*

**LEFT\_ALT***= u'\ue00a'*

**LEFT\_CONTROL***= u'\ue009'*

**LEFT\_SHIFT***= u'\ue008'*

**META***= u'\ue03d'*

**MULTIPLY***= u'\ue024'*

**NULL***= u'\ue000'*

**NUMPAD0***= u'\ue01a'*

**NUMPAD1***= u'\ue01b'*

**NUMPAD2***= u'\ue01c'*

**NUMPAD3***= u'\ue01d'*

**NUMPAD4***= u'\ue01e'*

**NUMPAD5***= u'\ue01f'*

**NUMPAD6***= u'\ue020'*

**NUMPAD7***= u'\ue021'*

**NUMPAD8***= u'\ue022'*

**NUMPAD9***= u'\ue023'*

**PAGE\_DOWN***= u'\ue00f'*

**PAGE\_UP***= u'\ue00e'*

**PAUSE***= u'\ue00b'*

**RETURN***= u'\ue006'*

**RIGHT***= u'\ue014'*

**SEMICOLON***= u'\ue018'*

**SEPARATOR***= u'\ue026'*

**SHIFT***= u'\ue008'*

**SPACE***= u'\ue00d'*

**SUBTRACT***= u'\ue027'*

**TAB***= u'\ue004'*

**UP***= u'\ue013'*

7.5. Locate elements By

These are the attributes which can be used to locate elements. See the [Locating Elements](http://selenium-python.readthedocs.io/locating-elements.html#locating-elements) chapter for example usages.

The By implementation.

*class*selenium.webdriver.common.by.**By**

Bases: **object**

Set of supported locator strategies.

**CLASS\_NAME***= 'class name'*

**CSS\_SELECTOR***= 'css selector'*

**ID***= 'id'*

**LINK\_TEXT***= 'link text'*

**NAME***= 'name'*

**PARTIAL\_LINK\_TEXT***= 'partial link text'*

**TAG\_NAME***= 'tag name'*

**XPATH***= 'xpath'*

7.6. Desired Capabilities

See the [Using Selenium with remote WebDriver](http://selenium-python.readthedocs.io/getting-started.html#selenium-remote-webdriver) section for example usages of desired capabilities.

The Desired Capabilities implementation.

*class*selenium.webdriver.common.desired\_capabilities.**DesiredCapabilities**

Bases: **object**

Set of default supported desired capabilities.

Use this as a starting point for creating a desired capabilities object for requesting remote webdrivers for connecting to selenium server or selenium grid.

Usage Example:

**from** **selenium** **import** webdriver

selenium\_grid\_url = "http://198.0.0.1:4444/wd/hub"

*# Create a desired capabilities object as a starting point.*

capabilities = DesiredCapabilities.FIREFOX.copy()

capabilities['platform'] = "WINDOWS"

capabilities['version'] = "10"

*# Instantiate an instance of Remote WebDriver with the desired capabilities.*

driver = webdriver.Remote(desired\_capabilities=capabilities,

command\_executor=selenium\_grid\_url)

Note: Always use ‘.copy()’ on the DesiredCapabilities object to avoid the side effects of altering the Global class instance.

**ANDROID***= {'platform': 'ANDROID', 'browserName': 'android', 'version': ''}*

**CHROME***= {'platform': 'ANY', 'browserName': 'chrome', 'version': ''}*

**EDGE***= {'platform': 'WINDOWS', 'browserName': 'MicrosoftEdge', 'version': ''}*

**FIREFOX***= {'acceptInsecureCerts': True, 'browserName': 'firefox', 'marionette': True}*

**HTMLUNIT***= {'platform': 'ANY', 'browserName': 'htmlunit', 'version': ''}*

**HTMLUNITWITHJS***= {'platform': 'ANY', 'browserName': 'htmlunit', 'version': 'firefox', 'javascriptEnabled': True}*

**INTERNETEXPLORER***= {'platform': 'WINDOWS', 'browserName': 'internet explorer', 'version': ''}*

**IPAD***= {'platform': 'MAC', 'browserName': 'iPad', 'version': ''}*

**IPHONE***= {'platform': 'MAC', 'browserName': 'iPhone', 'version': ''}*

**OPERA***= {'platform': 'ANY', 'browserName': 'opera', 'version': ''}*

**PHANTOMJS***= {'platform': 'ANY', 'browserName': 'phantomjs', 'version': '', 'javascriptEnabled': True}*

**SAFARI***= {'platform': 'MAC', 'browserName': 'safari', 'version': ''}*

7.7. Utilities

The Utils methods.

selenium.webdriver.common.utils.**find\_connectable\_ip**(*host*, *port=None*)

Resolve a hostname to an IP, preferring IPv4 addresses.

We prefer IPv4 so that we don’t change behavior from previous IPv4-only implementations, and because some drivers (e.g., FirefoxDriver) do not support IPv6 connections.

If the optional port number is provided, only IPs that listen on the given port are considered.

|  |  |
| --- | --- |
| **Args:** | * host - A hostname. * port - Optional port number. |
| **Returns:** | A single IP address, as a string. If any IPv4 address is found, one is returned. Otherwise, if any IPv6 address is found, one is returned. If neither, then None is returned. |

selenium.webdriver.common.utils.**free\_port**()

Determines a free port using sockets.

selenium.webdriver.common.utils.**is\_connectable**(*port*, *host='localhost'*)

Tries to connect to the server at port to see if it is running.

|  |  |
| --- | --- |
| **Args:** | * port - The port to connect. |

selenium.webdriver.common.utils.**is\_url\_connectable**(*port*)

Tries to connect to the HTTP server at /status path and specified port to see if it responds successfully.

|  |  |
| --- | --- |
| **Args:** | * port - The port to connect. |

selenium.webdriver.common.utils.**join\_host\_port**(*host*, *port*)

Joins a hostname and port together.

This is a minimal implementation intended to cope with IPv6 literals. For example, \_join\_host\_port(‘::1’, 80) == ‘[::1]:80’.

|  |  |
| --- | --- |
| **Args:** | * host - A hostname. * port - An integer port. |

selenium.webdriver.common.utils.**keys\_to\_typing**(*value*)

Processes the values that will be typed in the element.

7.8. Firefox WebDriver

*class*selenium.webdriver.firefox.webdriver.**WebDriver**(*firefox\_profile=None*, *firefox\_binary=None*, *timeout=30*, *capabilities=None*, *proxy=None*, *executable\_path='geckodriver'*, *firefox\_options=None*, *log\_path='geckodriver.log'*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](http://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

**context**(*\*args*, *\*\*kwds*)

Sets the context that Selenium commands are running in using a *with* statement. The state of the context on the server is saved before entering the block, and restored upon exiting it.

|  |  |
| --- | --- |
| **Parameters:** | **context** – Context, may be one of the class properties*CONTEXT\_CHROME* or *CONTEXT\_CONTENT*. |

Usage example:

**with** selenium.context(selenium.CONTEXT\_CHROME):

*# chrome scope*

... do stuff ...

**install\_addon**(*path*, *temporary=None*)

Installs Firefox addon.

Returns identifier of installed addon. This identifier can later be used to uninstall addon.

|  |  |
| --- | --- |
| **Usage:** | driver.install\_addon(‘firebug.xpi’) |

**quit**()

Quits the driver and close every associated window.

**set\_context**(*context*)

**uninstall\_addon**(*identifier*)

Uninstalls Firefox addon using its identifier.

|  |  |
| --- | --- |
| **Usage:** | driver.uninstall\_addon(['addon@foo.com](mailto:'addon%40foo.com)‘) |

**CONTEXT\_CHROME***= 'chrome'*

**CONTEXT\_CONTENT***= 'content'*

**NATIVE\_EVENTS\_ALLOWED***= True*

**firefox\_profile**

7.9. Chrome WebDriver

*class*selenium.webdriver.chrome.webdriver.**WebDriver**(*executable\_path='chromedriver'*, *port=0*, *chrome\_options=None*, *service\_args=None*, *desired\_capabilities=None*, *service\_log\_path=None*)

Bases: [**selenium.webdriver.remote.webdriver.WebDriver**](http://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webdriver.WebDriver)

Controls the ChromeDriver and allows you to drive the browser.

You will need to download the ChromeDriver executable from<http://chromedriver.storage.googleapis.com/index.html>

**create\_options**()

**get\_network\_conditions**()

Gets Chrome network emulation settings.

|  |  |
| --- | --- |
| **Returns:** | A dict. For example:  {‘latency’: 4, ‘download\_throughput’: 2, ‘upload\_throughput’: 2, ‘offline’: False} |

**launch\_app**(*id*)

Launches Chrome app specified by id.

**quit**()

Closes the browser and shuts down the ChromeDriver executable that is started when starting the ChromeDriver

**set\_network\_conditions**(*\*\*network\_conditions*)

Sets Chrome network emulation settings.

|  |  |
| --- | --- |
| **Args:** | * network\_conditions: A dict with conditions specification. |
| **Usage:** | driver.set\_network\_conditions(  offline=False, latency=5, # additional latency (ms) download\_throughput=500 \* 1024, # maximal throughput upload\_throughput=500 \* 1024) # maximal throughput  Note: ‘throughput’ can be used to set both (for download and upload). |

7.10. Remote WebDriver

The WebDriver implementation.

*class*selenium.webdriver.remote.webdriver.**WebDriver**(*command\_executor='http://127.0.0.1:4444/wd/hub'*, *desired\_capabilities=None*, *browser\_profile=None*, *proxy=None*, *keep\_alive=False*, *file\_detector=None*)

Bases: **object**

Controls a browser by sending commands to a remote server. This server is expected to be running the WebDriver wire protocol as defined at<https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol>

|  |  |
| --- | --- |
| **Attributes:** | * session\_id - String ID of the browser session started and controlled by this WebDriver. * capabilities - Dictionaty of effective capabilities of this browser session as returned   by the remote server. See <https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities>   * command\_executor - remote\_connection.RemoteConnection object used to execute commands. * error\_handler - errorhandler.ErrorHandler object used to handle errors. |

**add\_cookie**(*cookie\_dict*)

Adds a cookie to your current session.

|  |  |
| --- | --- |
| **Args:** | * cookie\_dict: A dictionary object, with required keys - “name” and “value”;   optional keys - “path”, “domain”, “secure”, “expiry” |

Usage:

driver.add\_cookie({‘name’ : ‘foo’, ‘value’ : ‘bar’}) driver.add\_cookie({‘name’ : ‘foo’, ‘value’ : ‘bar’, ‘path’ : ‘/’}) driver.add\_cookie({‘name’ : ‘foo’, ‘value’ : ‘bar’, ‘path’ : ‘/’, ‘secure’:True})

**back**()

Goes one step backward in the browser history.

|  |  |
| --- | --- |
| **Usage:** | driver.back() |

**close**()

Closes the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.close() |

**create\_web\_element**(*element\_id*)

Creates a web element with the specified *element\_id*.

**delete\_all\_cookies**()

Delete all cookies in the scope of the session.

|  |  |
| --- | --- |
| **Usage:** | driver.delete\_all\_cookies() |

**delete\_cookie**(*name*)

Deletes a single cookie with the given name.

|  |  |
| --- | --- |
| **Usage:** | driver.delete\_cookie(‘my\_cookie’) |

**execute**(*driver\_command*, *params=None*)

Sends a command to be executed by a command.CommandExecutor.

|  |  |
| --- | --- |
| **Args:** | * driver\_command: The name of the command to execute as a string. * params: A dictionary of named parameters to send with the command. |
| **Returns:** | The command’s JSON response loaded into a dictionary object. |

**execute\_async\_script**(*script*, *\*args*)

Asynchronously Executes JavaScript in the current window/frame.

|  |  |
| --- | --- |
| **Args:** | * script: The JavaScript to execute. * \*args: Any applicable arguments for your JavaScript. |
| **Usage:** | driver.execute\_async\_script(‘document.title’) |

**execute\_script**(*script*, *\*args*)

Synchronously Executes JavaScript in the current window/frame.

|  |  |
| --- | --- |
| **Args:** | * script: The JavaScript to execute. * \*args: Any applicable arguments for your JavaScript. |
| **Usage:** | driver.execute\_script(‘document.title’) |

**file\_detector\_context**(*\*args*, *\*\*kwds*)

Overrides the current file detector (if necessary) in limited context. Ensures the original file detector is set afterwards.

Example:

with webdriver.file\_detector\_context(UselessFileDetector):

someinput.send\_keys(‘/etc/hosts’)

|  |  |
| --- | --- |
| **Args:** | * file\_detector\_class - Class of the desired file detector. If the class is different   from the current file\_detector, then the class is instantiated with args and kwargs and used as a file detector during the duration of the context manager.   * args - Optional arguments that get passed to the file detector class during   instantiation.   * kwargs - Keyword arguments, passed the same way as args. |

**find\_element**(*by='id'*, *value=None*)

‘Private’ method used by the find\_element\_by\_\* methods.

|  |  |
| --- | --- |
| **Usage:** | Use the corresponding find\_element\_by\_\* instead of this. |
| **Return type:** | [WebElement](http://selenium-python.readthedocs.io/api.html#selenium.webdriver.remote.webelement.WebElement) |

**find\_element\_by\_class\_name**(*name*)

Finds an element by class name.

|  |  |
| --- | --- |
| **Args:** | * name: The class name of the element to find. |
| **Usage:** | driver.find\_element\_by\_class\_name(‘foo’) |

**find\_element\_by\_css\_selector**(*css\_selector*)

Finds an element by css selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector: The css selector to use when finding elements. |
| **Usage:** | driver.find\_element\_by\_css\_selector(‘#foo’) |

**find\_element\_by\_id**(*id\_*)

Finds an element by id.

|  |  |
| --- | --- |
| **Args:** | * id\_ - The id of the element to be found. |
| **Usage:** | driver.find\_element\_by\_id(‘foo’) |

**find\_element\_by\_link\_text**(*link\_text*)

Finds an element by link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to be found. |
| **Usage:** | driver.find\_element\_by\_link\_text(‘Sign In’) |

**find\_element\_by\_name**(*name*)

Finds an element by name.

|  |  |
| --- | --- |
| **Args:** | * name: The name of the element to find. |
| **Usage:** | driver.find\_element\_by\_name(‘foo’) |

**find\_element\_by\_partial\_link\_text**(*link\_text*)

Finds an element by a partial match of its link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to partially match on. |
| **Usage:** | driver.find\_element\_by\_partial\_link\_text(‘Sign’) |

**find\_element\_by\_tag\_name**(*name*)

Finds an element by tag name.

|  |  |
| --- | --- |
| **Args:** | * name: The tag name of the element to find. |
| **Usage:** | driver.find\_element\_by\_tag\_name(‘foo’) |

**find\_element\_by\_xpath**(*xpath*)

Finds an element by xpath.

|  |  |
| --- | --- |
| **Args:** | * xpath - The xpath locator of the element to find. |
| **Usage:** | driver.find\_element\_by\_xpath(‘//div/td[1]’) |

**find\_elements**(*by='id'*, *value=None*)

‘Private’ method used by the find\_elements\_by\_\* methods.

|  |  |
| --- | --- |
| **Usage:** | Use the corresponding find\_elements\_by\_\* instead of this. |
| **Return type:** | list of WebElement |

**find\_elements\_by\_class\_name**(*name*)

Finds elements by class name.

|  |  |
| --- | --- |
| **Args:** | * name: The class name of the elements to find. |
| **Usage:** | driver.find\_elements\_by\_class\_name(‘foo’) |

**find\_elements\_by\_css\_selector**(*css\_selector*)

Finds elements by css selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector: The css selector to use when finding elements. |
| **Usage:** | driver.find\_elements\_by\_css\_selector(‘.foo’) |

**find\_elements\_by\_id**(*id\_*)

Finds multiple elements by id.

|  |  |
| --- | --- |
| **Args:** | * id\_ - The id of the elements to be found. |
| **Usage:** | driver.find\_elements\_by\_id(‘foo’) |

**find\_elements\_by\_link\_text**(*text*)

Finds elements by link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the elements to be found. |
| **Usage:** | driver.find\_elements\_by\_link\_text(‘Sign In’) |

**find\_elements\_by\_name**(*name*)

Finds elements by name.

|  |  |
| --- | --- |
| **Args:** | * name: The name of the elements to find. |
| **Usage:** | driver.find\_elements\_by\_name(‘foo’) |

**find\_elements\_by\_partial\_link\_text**(*link\_text*)

Finds elements by a partial match of their link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text: The text of the element to partial match on. |
| **Usage:** | driver.find\_element\_by\_partial\_link\_text(‘Sign’) |

**find\_elements\_by\_tag\_name**(*name*)

Finds elements by tag name.

|  |  |
| --- | --- |
| **Args:** | * name: The tag name the use when finding elements. |
| **Usage:** | driver.find\_elements\_by\_tag\_name(‘foo’) |

**find\_elements\_by\_xpath**(*xpath*)

Finds multiple elements by xpath.

|  |  |
| --- | --- |
| **Args:** | * xpath - The xpath locator of the elements to be found. |
| **Usage:** | driver.find\_elements\_by\_xpath(“//div[contains(@class, ‘foo’)]”) |

**forward**()

Goes one step forward in the browser history.

|  |  |
| --- | --- |
| **Usage:** | driver.forward() |

**get**(*url*)

Loads a web page in the current browser session.

**get\_cookie**(*name*)

Get a single cookie by name. Returns the cookie if found, None if not.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_cookie(‘my\_cookie’) |

**get\_cookies**()

Returns a set of dictionaries, corresponding to cookies visible in the current session.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_cookies() |

**get\_log**(*log\_type*)

Gets the log for a given log type

|  |  |
| --- | --- |
| **Args:** | * log\_type: type of log that which will be returned |
| **Usage:** | driver.get\_log(‘browser’) driver.get\_log(‘driver’) driver.get\_log(‘client’) driver.get\_log(‘server’) |

**get\_screenshot\_as\_base64**()

Gets the screenshot of the current window as a base64 encoded string

which is useful in embedded images in HTML.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_screenshot\_as\_base64() |

**get\_screenshot\_as\_file**(*filename*)

Saves a screenshot of the current window to a PNG image file. Returns

False if there is any IOError, else returns True. Use full paths in your filename.

|  |  |
| --- | --- |
| **Args:** | * filename: The full path you wish to save your screenshot to. This should end with a *.png* extension. |
| **Usage:** | driver.get\_screenshot\_as\_file(‘/Screenshots/foo.png’) |

**get\_screenshot\_as\_png**()

Gets the screenshot of the current window as a binary data.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_screenshot\_as\_png() |

**get\_window\_position**(*windowHandle='current'*)

Gets the x,y position of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_window\_position() |

**get\_window\_rect**()

Gets the x, y coordinates of the window as well as height and width of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_window\_rect() |

**get\_window\_size**(*windowHandle='current'*)

Gets the width and height of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.get\_window\_size() |

**implicitly\_wait**(*time\_to\_wait*)

Sets a sticky timeout to implicitly wait for an element to be found,

or a command to complete. This method only needs to be called one time per session. To set the timeout for calls to execute\_async\_script, see set\_script\_timeout.

|  |  |
| --- | --- |
| **Args:** | * time\_to\_wait: Amount of time to wait (in seconds) |
| **Usage:** | driver.implicitly\_wait(30) |

**maximize\_window**()

Maximizes the current window that webdriver is using

**quit**()

Quits the driver and closes every associated window.

|  |  |
| --- | --- |
| **Usage:** | driver.quit() |

**refresh**()

Refreshes the current page.

|  |  |
| --- | --- |
| **Usage:** | driver.refresh() |

**save\_screenshot**(*filename*)

Saves a screenshot of the current window to a PNG image file. Returns

False if there is any IOError, else returns True. Use full paths in your filename.

|  |  |
| --- | --- |
| **Args:** | * filename: The full path you wish to save your screenshot to. This should end with a *.png* extension. |
| **Usage:** | driver.save\_screenshot(‘/Screenshots/foo.png’) |

**set\_page\_load\_timeout**(*time\_to\_wait*)

Set the amount of time to wait for a page load to complete

before throwing an error.

|  |  |
| --- | --- |
| **Args:** | * time\_to\_wait: The amount of time to wait |
| **Usage:** | driver.set\_page\_load\_timeout(30) |

**set\_script\_timeout**(*time\_to\_wait*)

Set the amount of time that the script should wait during an

execute\_async\_script call before throwing an error.

|  |  |
| --- | --- |
| **Args:** | * time\_to\_wait: The amount of time to wait (in seconds) |
| **Usage:** | driver.set\_script\_timeout(30) |

**set\_window\_position**(*x*, *y*, *windowHandle='current'*)

Sets the x,y position of the current window. (window.moveTo)

|  |  |
| --- | --- |
| **Args:** | * x: the x-coordinate in pixels to set the window position * y: the y-coordinate in pixels to set the window position |
| **Usage:** | driver.set\_window\_position(0,0) |

**set\_window\_rect**(*x=None*, *y=None*, *width=None*, *height=None*)

Sets the x, y coordinates of the window as well as height and width of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.set\_window\_rect(x=10, y=10) driver.set\_window\_rect(width=100, height=200) driver.set\_window\_rect(x=10, y=10, width=100, height=200) |

**set\_window\_size**(*width*, *height*, *windowHandle='current'*)

Sets the width and height of the current window. (window.resizeTo)

|  |  |
| --- | --- |
| **Args:** | * width: the width in pixels to set the window to * height: the height in pixels to set the window to |
| **Usage:** | driver.set\_window\_size(800,600) |

**start\_client**()

Called before starting a new session. This method may be overridden to define custom startup behavior.

**start\_session**(*capabilities*, *browser\_profile=None*)

Creates a new session with the desired capabilities.

|  |  |
| --- | --- |
| **Args:** | * browser\_name - The name of the browser to request. * version - Which browser version to request. * platform - Which platform to request the browser on. * javascript\_enabled - Whether the new session should support JavaScript. * browser\_profile - A selenium.webdriver.firefox.firefox\_profile.FirefoxProfile object. Only used if Firefox is requested. |

**stop\_client**()

Called after executing a quit command. This method may be overridden to define custom shutdown behavior.

**switch\_to\_active\_element**()

Deprecated use driver.switch\_to.active\_element

**switch\_to\_alert**()

Deprecated use driver.switch\_to.alert

**switch\_to\_default\_content**()

Deprecated use driver.switch\_to.default\_content

**switch\_to\_frame**(*frame\_reference*)

Deprecated use driver.switch\_to.frame

**switch\_to\_window**(*window\_name*)

Deprecated use driver.switch\_to.window

**application\_cache**

Returns a ApplicationCache Object to interact with the browser app cache

**current\_url**

Gets the URL of the current page.

|  |  |
| --- | --- |
| **Usage:** | driver.current\_url |

**current\_window\_handle**

Returns the handle of the current window.

|  |  |
| --- | --- |
| **Usage:** | driver.current\_window\_handle |

**desired\_capabilities**

returns the drivers current desired capabilities being used

**file\_detector**

**log\_types**

Gets a list of the available log types

|  |  |
| --- | --- |
| **Usage:** | driver.log\_types |

**mobile**

**name**

Returns the name of the underlying browser for this instance.

|  |  |
| --- | --- |
| **Usage:** | * driver.name |

**orientation**

Gets the current orientation of the device

|  |  |
| --- | --- |
| **Usage:** | orientation = driver.orientation |

**page\_source**

Gets the source of the current page.

|  |  |
| --- | --- |
| **Usage:** | driver.page\_source |

**switch\_to**

**title**

Returns the title of the current page.

|  |  |
| --- | --- |
| **Usage:** | driver.title |

**window\_handles**

Returns the handles of all windows within the current session.

|  |  |
| --- | --- |
| **Usage:** | driver.window\_handles |

7.11. WebElement

*class*selenium.webdriver.remote.webelement.**WebElement**(*parent*, *id\_*, *w3c=False*)

Bases: **object**

Represents a DOM element.

Generally, all interesting operations that interact with a document will be performed through this interface.

All method calls will do a freshness check to ensure that the element reference is still valid. This essentially determines whether or not the element is still attached to the DOM. If this test fails, then an StaleElementReferenceException is thrown, and all future calls to this instance will fail.

**clear**()

Clears the text if it’s a text entry element.

**click**()

Clicks the element.

**find\_element**(*by='id'*, *value=None*)

**find\_element\_by\_class\_name**(*name*)

Finds element within this element’s children by class name.

|  |  |
| --- | --- |
| **Args:** | * name - class name to search for. |

**find\_element\_by\_css\_selector**(*css\_selector*)

Finds element within this element’s children by CSS selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector - CSS selctor string, ex: ‘a.nav#home’ |

**find\_element\_by\_id**(*id\_*)

Finds element within this element’s children by ID.

|  |  |
| --- | --- |
| **Args:** | * id\_ - ID of child element to locate. |

**find\_element\_by\_link\_text**(*link\_text*)

Finds element within this element’s children by visible link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text - Link text string to search for. |

**find\_element\_by\_name**(*name*)

Finds element within this element’s children by name.

|  |  |
| --- | --- |
| **Args:** | * name - name property of the element to find. |

**find\_element\_by\_partial\_link\_text**(*link\_text*)

Finds element within this element’s children by partially visible link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text - Link text string to search for. |

**find\_element\_by\_tag\_name**(*name*)

Finds element within this element’s children by tag name.

|  |  |
| --- | --- |
| **Args:** | * name - name of html tag (eg: h1, a, span) |

**find\_element\_by\_xpath**(*xpath*)

Finds element by xpath.

|  |  |
| --- | --- |
| **Args:** | xpath - xpath of element to locate. “//input[@class=’myelement’]” |

Note: The base path will be relative to this element’s location.

This will select the first link under this element.

myelement.find\_elements\_by\_xpath(".//a")

However, this will select the first link on the page.

myelement.find\_elements\_by\_xpath("//a")

**find\_elements**(*by='id'*, *value=None*)

**find\_elements\_by\_class\_name**(*name*)

Finds a list of elements within this element’s children by class name.

|  |  |
| --- | --- |
| **Args:** | * name - class name to search for. |

**find\_elements\_by\_css\_selector**(*css\_selector*)

Finds a list of elements within this element’s children by CSS selector.

|  |  |
| --- | --- |
| **Args:** | * css\_selector - CSS selctor string, ex: ‘a.nav#home’ |

**find\_elements\_by\_id**(*id\_*)

Finds a list of elements within this element’s children by ID.

|  |  |
| --- | --- |
| **Args:** | * id\_ - Id of child element to find. |

**find\_elements\_by\_link\_text**(*link\_text*)

Finds a list of elements within this element’s children by visible link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text - Link text string to search for. |

**find\_elements\_by\_name**(*name*)

Finds a list of elements within this element’s children by name.

|  |  |
| --- | --- |
| **Args:** | * name - name property to search for. |

**find\_elements\_by\_partial\_link\_text**(*link\_text*)

Finds a list of elements within this element’s children by link text.

|  |  |
| --- | --- |
| **Args:** | * link\_text - Link text string to search for. |

**find\_elements\_by\_tag\_name**(*name*)

Finds a list of elements within this element’s children by tag name.

|  |  |
| --- | --- |
| **Args:** | * name - name of html tag (eg: h1, a, span) |

**find\_elements\_by\_xpath**(*xpath*)

Finds elements within the element by xpath.

|  |  |
| --- | --- |
| **Args:** | * xpath - xpath locator string. |

Note: The base path will be relative to this element’s location.

This will select all links under this element.

myelement.find\_elements\_by\_xpath(".//a")

However, this will select all links in the page itself.

myelement.find\_elements\_by\_xpath("//a")

**get\_attribute**(*name*)

Gets the given attribute or property of the element.

This method will first try to return the value of a property with the given name. If a property with that name doesn’t exist, it returns the value of the attribute with the same name. If there’s no attribute with that name, None is returned.

Values which are considered truthy, that is equals “true” or “false”, are returned as booleans. All other non-None values are returned as strings. For attributes or properties which do not exist, None is returned.

|  |  |
| --- | --- |
| **Args:** | * name - Name of the attribute/property to retrieve. |

Example:

*# Check if the "active" CSS class is applied to an element.*

is\_active = "active" **in** target\_element.get\_attribute("class")

**get\_property**(*name*)

Gets the given property of the element.

|  |  |
| --- | --- |
| **Args:** | * name - Name of the property to retrieve. |

Example:

*# Check if the "active" CSS class is applied to an element.*

text\_length = target\_element.get\_property("text\_length")

**is\_displayed**()

Whether the element is visible to a user.

**is\_enabled**()

Returns whether the element is enabled.

**is\_selected**()

Returns whether the element is selected.

Can be used to check if a checkbox or radio button is selected.

**screenshot**(*filename*)

Saves a screenshot of the current element to a PNG image file. Returns

False if there is any IOError, else returns True. Use full paths in your filename.

|  |  |
| --- | --- |
| **Args:** | * filename: The full path you wish to save your screenshot to. This should end with a *.png* extension. |
| **Usage:** | element.screenshot(‘/Screenshots/foo.png’) |

**send\_keys**(*\*value*)

Simulates typing into the element.

|  |  |
| --- | --- |
| **Args:** | * value - A string for typing, or setting form fields. For setting file inputs, this could be a local file path. |

Use this to send simple key events or to fill out form fields:

form\_textfield = driver.find\_element\_by\_name('username')

form\_textfield.send\_keys("admin")

This can also be used to set file inputs.

file\_input = driver.find\_element\_by\_name('profilePic')

file\_input.send\_keys("path/to/profilepic.gif")

*# Generally it's better to wrap the file path in one of the methods*

*# in os.path to return the actual path to support cross OS testing.*

*# file\_input.send\_keys(os.path.abspath("path/to/profilepic.gif"))*

**submit**()

Submits a form.

**value\_of\_css\_property**(*property\_name*)

The value of a CSS property.

**id**

Internal ID used by selenium.

This is mainly for internal use. Simple use cases such as checking if 2 webelements refer to the same element, can be done using ==:

**if** element1 == element2:

print("These 2 are equal")

**location**

The location of the element in the renderable canvas.

**location\_once\_scrolled\_into\_view**

THIS PROPERTY MAY CHANGE WITHOUT WARNING. Use this to discover where on the screen an element is so that we can click it. This method should cause the element to be scrolled into view.

Returns the top lefthand corner location on the screen, or None if the element is not visible.

**parent**

Internal reference to the WebDriver instance this element was found from.

**rect**

A dictionary with the size and location of the element.

**screenshot\_as\_base64**

Gets the screenshot of the current element as a base64 encoded string.

|  |  |
| --- | --- |
| **Usage:** | img\_b64 = element.screenshot\_as\_base64 |

**screenshot\_as\_png**

Gets the screenshot of the current element as a binary data.

|  |  |
| --- | --- |
| **Usage:** | element\_png = element.screenshot\_as\_png |

**size**

The size of the element.

**tag\_name**

This element’s tagName property.

**text**

The text of the element.

7.12. UI Support

*class*selenium.webdriver.support.select.**Select**(*webelement*)

Bases: **object**

**deselect\_all**()

Clear all selected entries. This is only valid when the SELECT supports multiple selections. throws NotImplementedError If the SELECT does not support multiple selections

**deselect\_by\_index**(*index*)

Deselect the option at the given index. This is done by examing the “index” attribute of an element, and not merely by counting.

|  |  |
| --- | --- |
| **Args:** | * index - The option at this index will be deselected   throws NoSuchElementException If there is no option with specisied index in SELECT |

**deselect\_by\_value**(*value*)

Deselect all options that have a value matching the argument. That is, when given “foo” this would deselect an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * value - The value to match against   throws NoSuchElementException If there is no option with specisied value in SELECT |

**deselect\_by\_visible\_text**(*text*)

Deselect all options that display text matching the argument. That is, when given “Bar” this would deselect an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * text - The visible text to match against |

**select\_by\_index**(*index*)

Select the option at the given index. This is done by examing the “index” attribute of an element, and not merely by counting.

|  |  |
| --- | --- |
| **Args:** | * index - The option at this index will be selected |

throws NoSuchElementException If there is no option with specisied index in SELECT

**select\_by\_value**(*value*)

Select all options that have a value matching the argument. That is, when given “foo” this would select an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * value - The value to match against |

throws NoSuchElementException If there is no option with specisied value in SELECT

**select\_by\_visible\_text**(*text*)

Select all options that display text matching the argument. That is, when given “Bar” this would select an option like:

<option value=”foo”>Bar</option>

|  |  |
| --- | --- |
| **Args:** | * text - The visible text to match against   throws NoSuchElementException If there is no option with specisied text in SELECT |

**all\_selected\_options**

Returns a list of all selected options belonging to this select tag

**first\_selected\_option**

The first selected option in this select tag (or the currently selected option in a normal select)

**options**

Returns a list of all options belonging to this select tag

*class*selenium.webdriver.support.wait.**WebDriverWait**(*driver*, *timeout*, *poll\_frequency=0.5*, *ignored\_exceptions=None*)

Bases: **object**

**until**(*method*, *message=''*)

Calls the method provided with the driver as an argument until the return value is not False.

**until\_not**(*method*, *message=''*)

Calls the method provided with the driver as an argument until the return value is False.

7.13. Color Support

*class*selenium.webdriver.support.color.**Color**(*red*, *green*, *blue*, *alpha=1*)

Bases: **object**

Color conversion support class

Example:

**from** **selenium.webdriver.support.color** **import** Color

**print**(Color.from\_string('#00ff33').rgba)

**print**(Color.from\_string('rgb(1, 255, 3)').hex)

**print**(Color.from\_string('blue').rgba)

*static***from\_string**(*str\_*)

**hex**

**rgb**

**rgba**

7.14. Expected conditions Support

*class*selenium.webdriver.support.expected\_conditions.**alert\_is\_present**

Bases: **object**

Expect an alert to be present.

*class*selenium.webdriver.support.expected\_conditions.**element\_located\_selection\_state\_to\_be**(*locator*, *is\_selected*)

Bases: **object**

An expectation to locate an element and check if the selection state specified is in that state. locator is a tuple of (by, path) is\_selected is a boolean

*class*selenium.webdriver.support.expected\_conditions.**element\_located\_to\_be\_selected**(*locator*)

Bases: **object**

An expectation for the element to be located is selected. locator is a tuple of (by, path)

*class*selenium.webdriver.support.expected\_conditions.**element\_selection\_state\_to\_be**(*element*, *is\_selected*)

Bases: **object**

An expectation for checking if the given element is selected. element is WebElement object is\_selected is a Boolean.”

*class*selenium.webdriver.support.expected\_conditions.**element\_to\_be\_clickable**(*locator*)

Bases: **object**

An Expectation for checking an element is visible and enabled such that you can click it.

*class*selenium.webdriver.support.expected\_conditions.**element\_to\_be\_selected**(*element*)

Bases: **object**

An expectation for checking the selection is selected. element is WebElement object

*class*selenium.webdriver.support.expected\_conditions.**frame\_to\_be\_available\_and\_switch\_to\_it**(*locator*)

Bases: **object**

An expectation for checking whether the given frame is available to switch to. If the frame is available it switches the given driver to the specified frame.

*class*selenium.webdriver.support.expected\_conditions.**invisibility\_of\_element\_located**(*locator*)

Bases: **object**

An Expectation for checking that an element is either invisible or not present on the DOM.

locator used to find the element

*class*selenium.webdriver.support.expected\_conditions.**new\_window\_is\_opened**(*current\_handles*)

Bases: **object**

An expectation that a new window will be opened and have the number of windows handles increase

*class*selenium.webdriver.support.expected\_conditions.**number\_of\_windows\_to\_be**(*num\_windows*)

Bases: **object**

An expectation for the number of windows to be a certain value.

*class*selenium.webdriver.support.expected\_conditions.**presence\_of\_all\_elements\_located**(*locator*)

Bases: **object**

An expectation for checking that there is at least one element present on a web page. locator is used to find the element returns the list of WebElements once they are located

*class*selenium.webdriver.support.expected\_conditions.**presence\_of\_element\_located**(*locator*)

Bases: **object**

An expectation for checking that an element is present on the DOM of a page. This does not necessarily mean that the element is visible. locator - used to find the element returns the WebElement once it is located

*class*selenium.webdriver.support.expected\_conditions.**staleness\_of**(*element*)

Bases: **object**

Wait until an element is no longer attached to the DOM. element is the element to wait for. returns False if the element is still attached to the DOM, true otherwise.

*class*selenium.webdriver.support.expected\_conditions.**text\_to\_be\_present\_in\_element**(*locator*, *text\_*)

Bases: **object**

An expectation for checking if the given text is present in the specified element. locator, text

*class*selenium.webdriver.support.expected\_conditions.**text\_to\_be\_present\_in\_element\_value**(*locator*, *text\_*)

Bases: **object**

An expectation for checking if the given text is present in the element’s locator, text

*class*selenium.webdriver.support.expected\_conditions.**title\_contains**(*title*)

Bases: **object**

An expectation for checking that the title contains a case-sensitive substring. title is the fragment of title expected returns True when the title matches, False otherwise

*class*selenium.webdriver.support.expected\_conditions.**title\_is**(*title*)

Bases: **object**

An expectation for checking the title of a page. title is the expected title, which must be an exact match returns True if the title matches, false otherwise.

*class*selenium.webdriver.support.expected\_conditions.**url\_changes**(*url*)

Bases: **object**

An expectation for checking the current url. url is the expected url, which must not be an exact match returns True if the url is different, false otherwise.

*class*selenium.webdriver.support.expected\_conditions.**url\_contains**(*url*)

Bases: **object**

An expectation for checking that the current url contains a case-sensitive substring. url is the fragment of url expected, returns True when the title matches, False otherwise

*class*selenium.webdriver.support.expected\_conditions.**url\_matches**(*pattern*)

Bases: **object**

An expectation for checking the current url. pattern is the expected pattern, which must be an exact match returns True if the title matches, false otherwise.

*class*selenium.webdriver.support.expected\_conditions.**url\_to\_be**(*url*)

Bases: **object**

An expectation for checking the current url. url is the expected url, which must be an exact match returns True if the title matches, false otherwise.

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of**(*element*)

Bases: **object**

An expectation for checking that an element, known to be present on the DOM of a page, is visible. Visibility means that the element is not only displayed but also has a height and width that is greater than 0. element is the WebElement returns the (same) WebElement once it is visible

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of\_all\_elements\_located**(*locator*)

Bases: **object**

An expectation for checking that all elements are present on the DOM of a page and visible. Visibility means that the elements are not only displayed but also has a height and width that is greater than 0. locator - used to find the elements returns the list of WebElements once they are located and visible

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of\_any\_elements\_located**(*locator*)

Bases: **object**

An expectation for checking that there is at least one element visible on a web page. locator is used to find the element returns the list of WebElements once they are located

*class*selenium.webdriver.support.expected\_conditions.**visibility\_of\_element\_located**(*locator*)

Bases: **object**

An expectation for checking that an element is present on the DOM of a page and visible. Visibility means that the element is not only displayed but also has a height and width that is greater than 0. locator - used to find the element returns the WebElement once it is located and visible